Architectures

Distributed Systems [2]

Distributed System Architecture

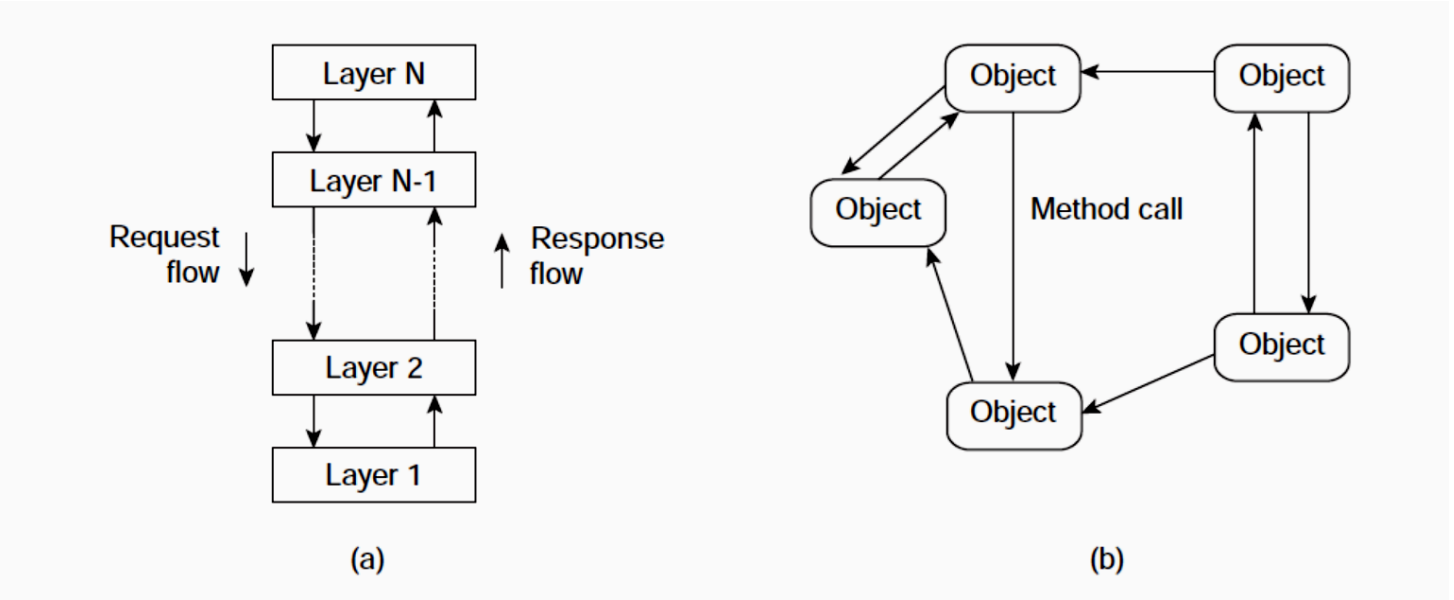
- A ***distributed system*** is many cooperating computers that appear to users as a single service.

- Distributed systems are often complex pieces of software of which the components are by definition dispersed across multiple machines.

- The organization of distributed systems is mostly about the software components that constitute the system.

Architectural Styles

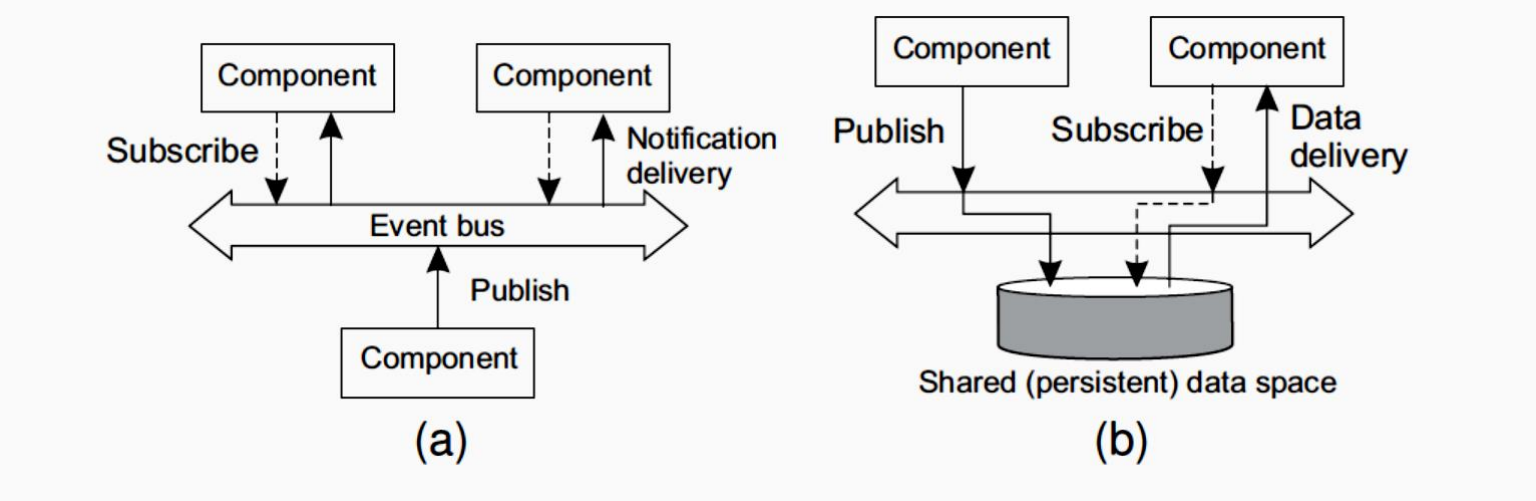
- Organize into logically different components, and distribute those components over the various machines.



(a) Layered style is used for client-server system.
(b) Object-based style for distributed object systems.

Architectural Styles

- Decoupling processes in space ("anonymous") and also time ("asynchronous") has led to alternative styles.



(a) Publish/subscribe [decoupled in **space**]
(b) Shared dataspace [decoupled in **space** and **time**]

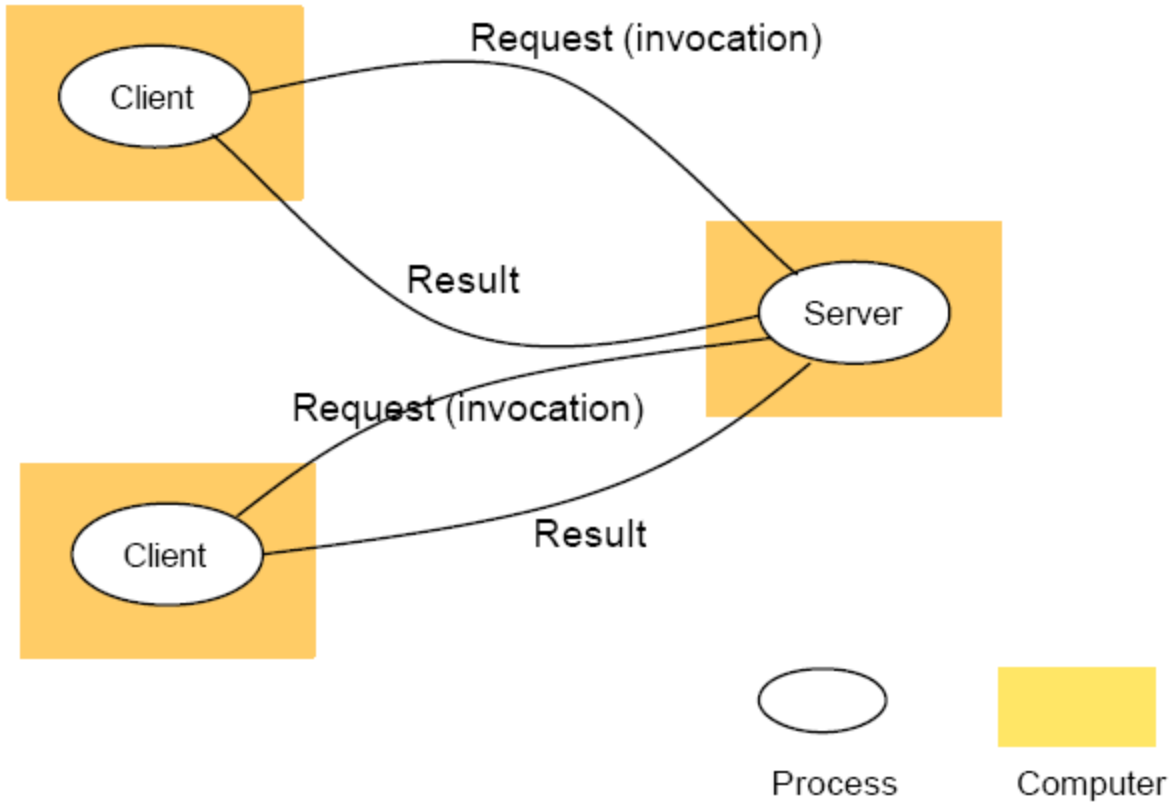
Organization

- Centralized
- Decentralized
- Hybrid

Centralized Architecture

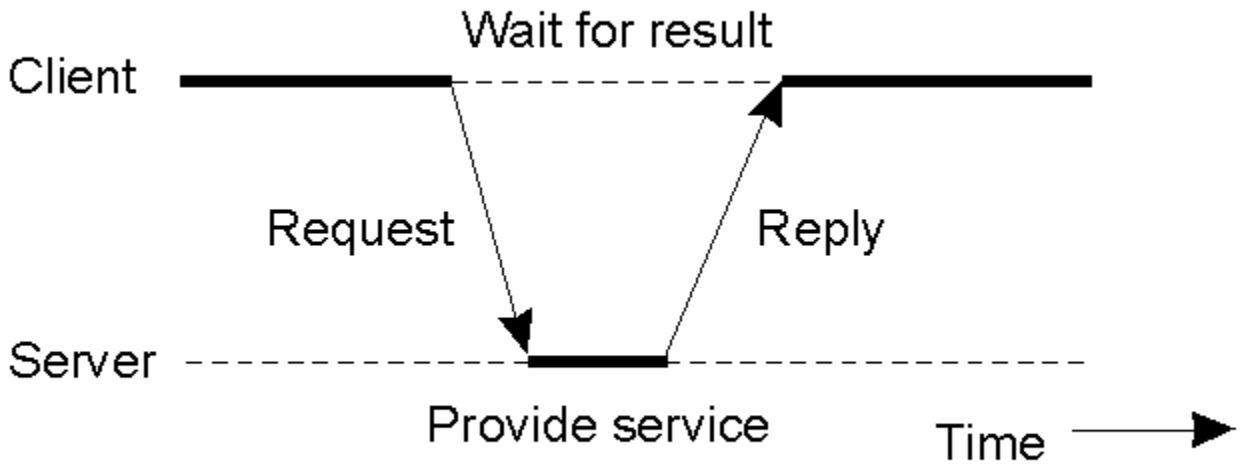
- **Basic Client–Server Model**
- Characteristics:
  - There are processes offering services (servers)
  - There are processes that use services (clients)
  - Clients and servers can be on different machines
  - Clients follow request/reply model wrt to using services

Client-Server Communication



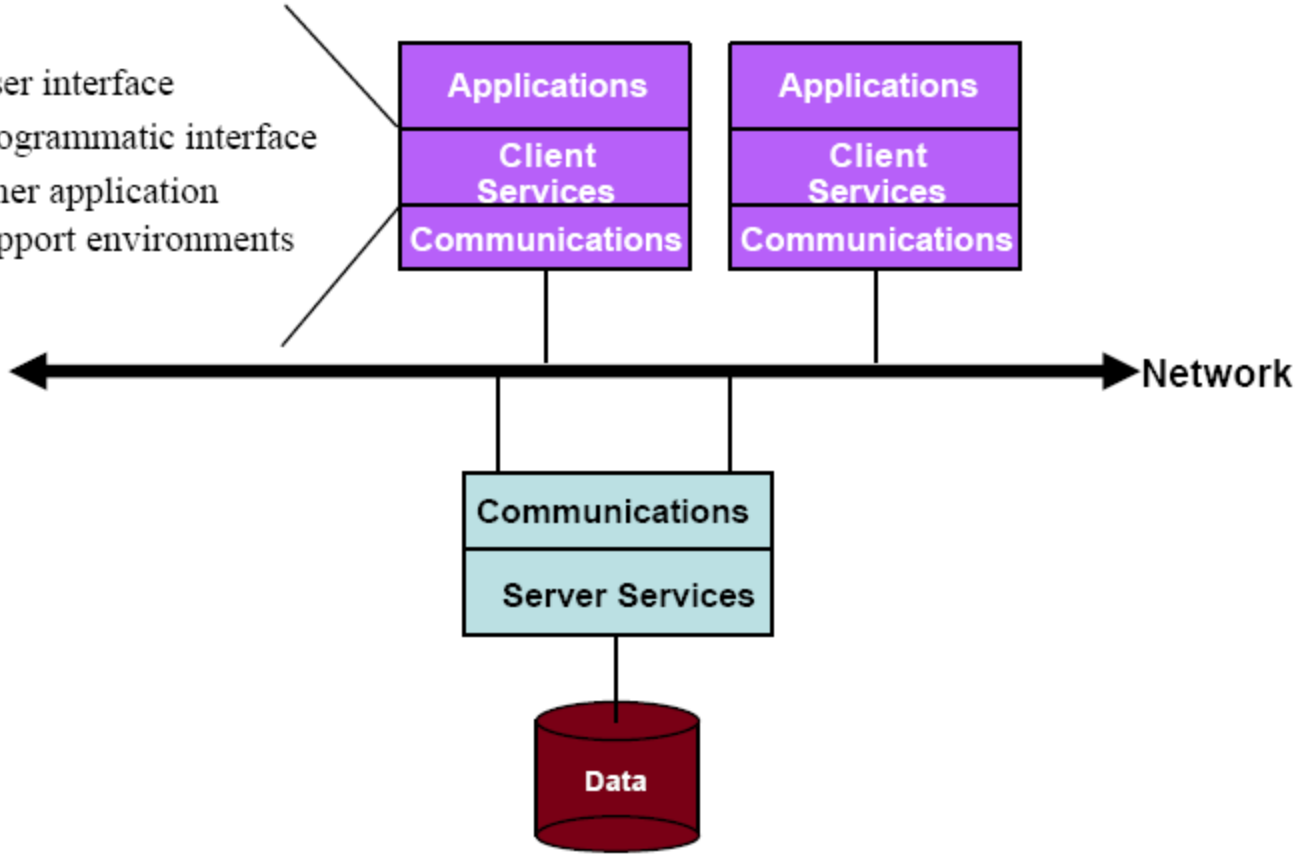
Clients and Servers, Timing

- General interaction between a client and a server.



Multiple-Client/Single Server

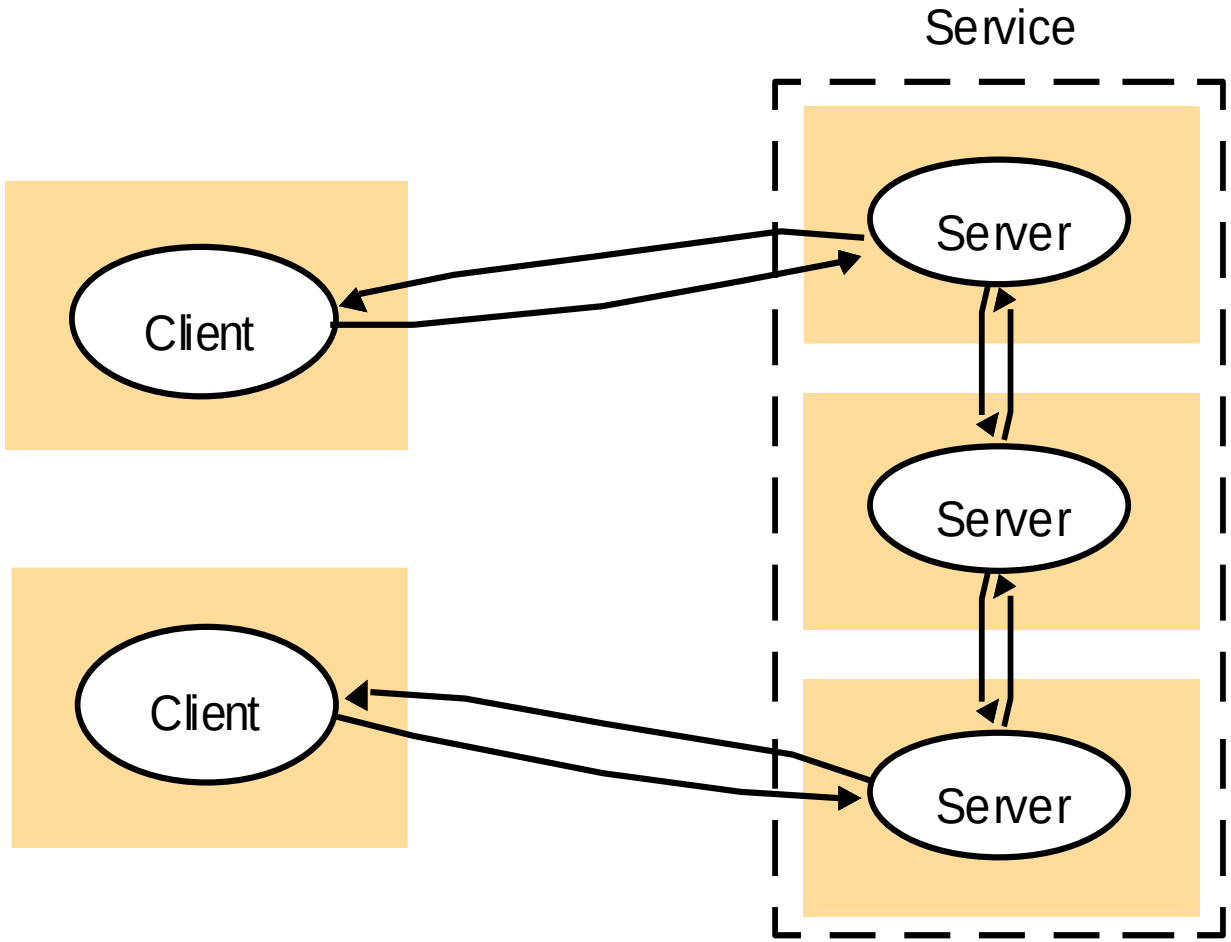
- User interface
- Programmatic interface
- other application support environments



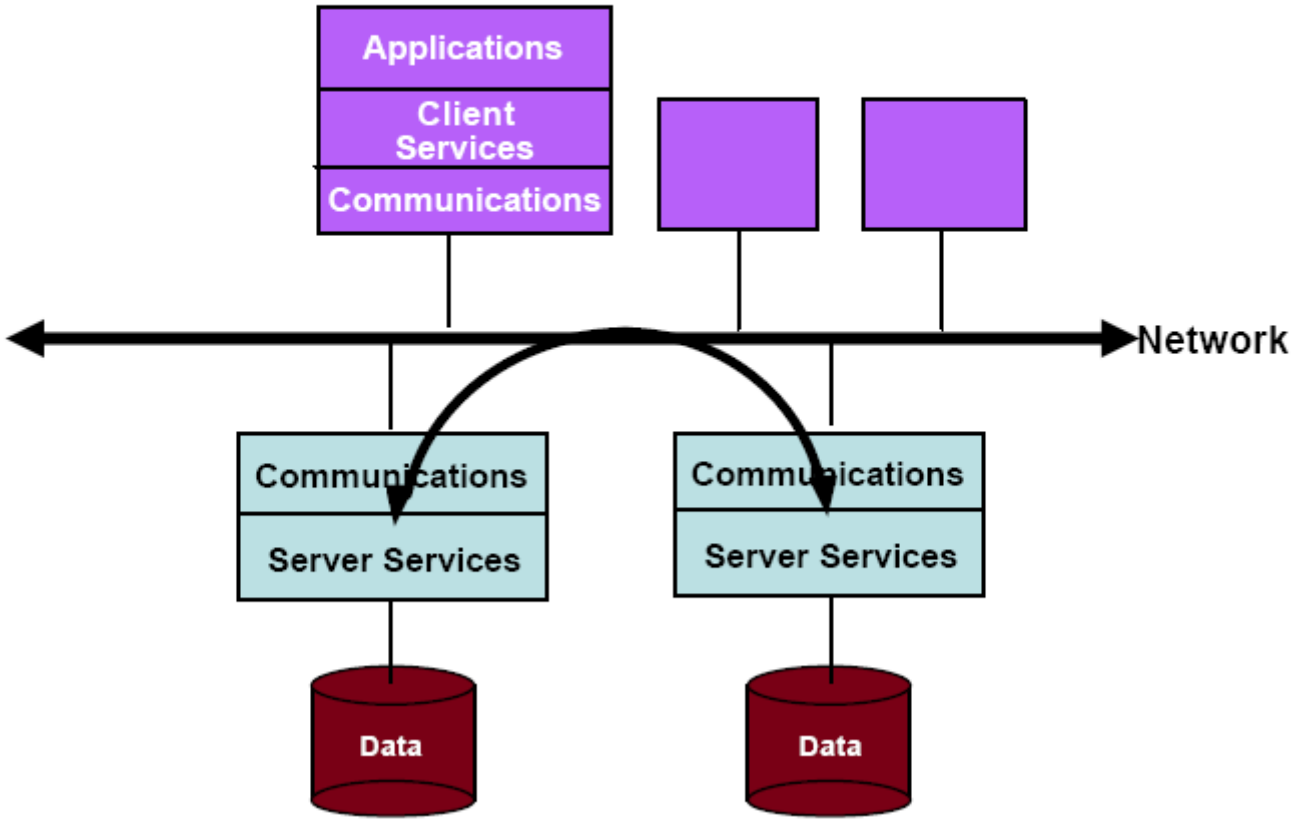
Problems With Multiple-Client/Single Server

- Server forms bottleneck
- Server forms single point of failure
- System scaling difficult

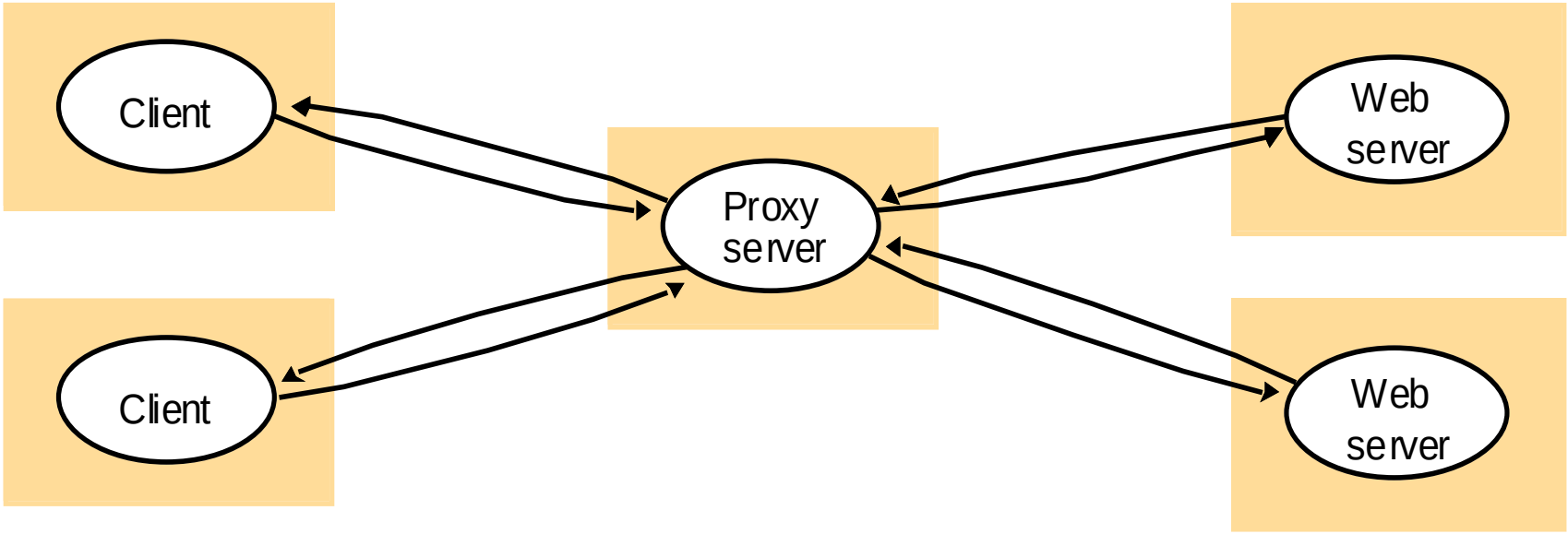
A service provided by multiple servers



Multiple Clients/Multiple Servers

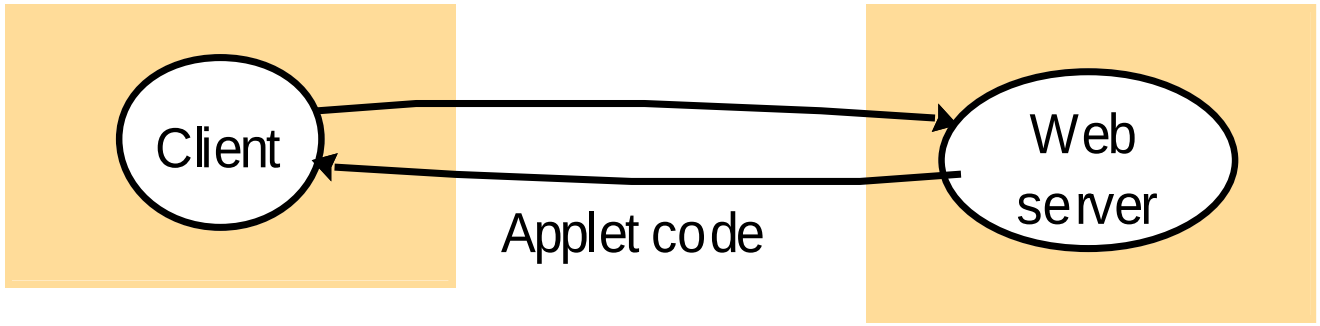


Web proxy server



Web Applets

a) client request results in the downloading of applet code

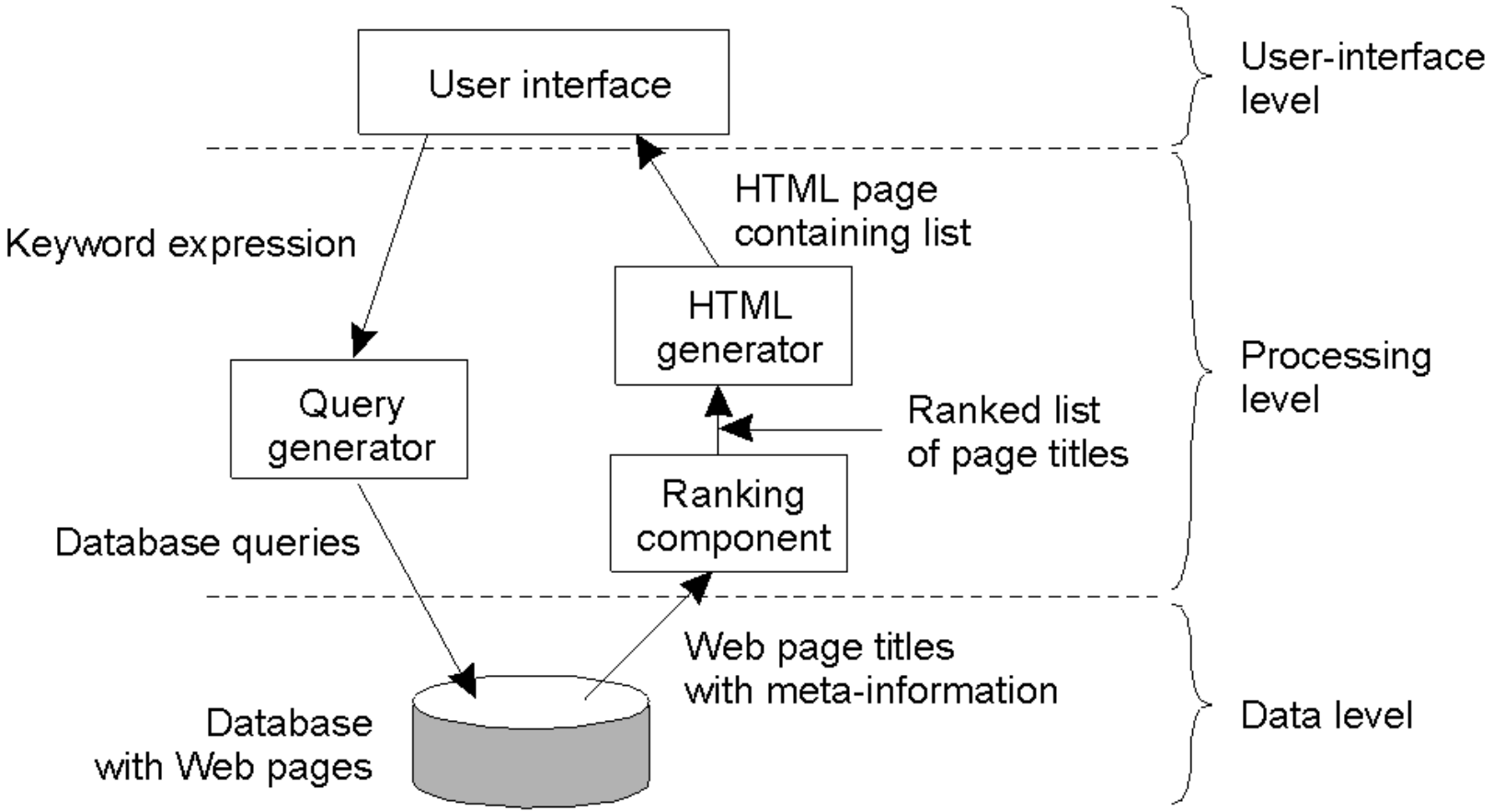


b) client interacts with the applet

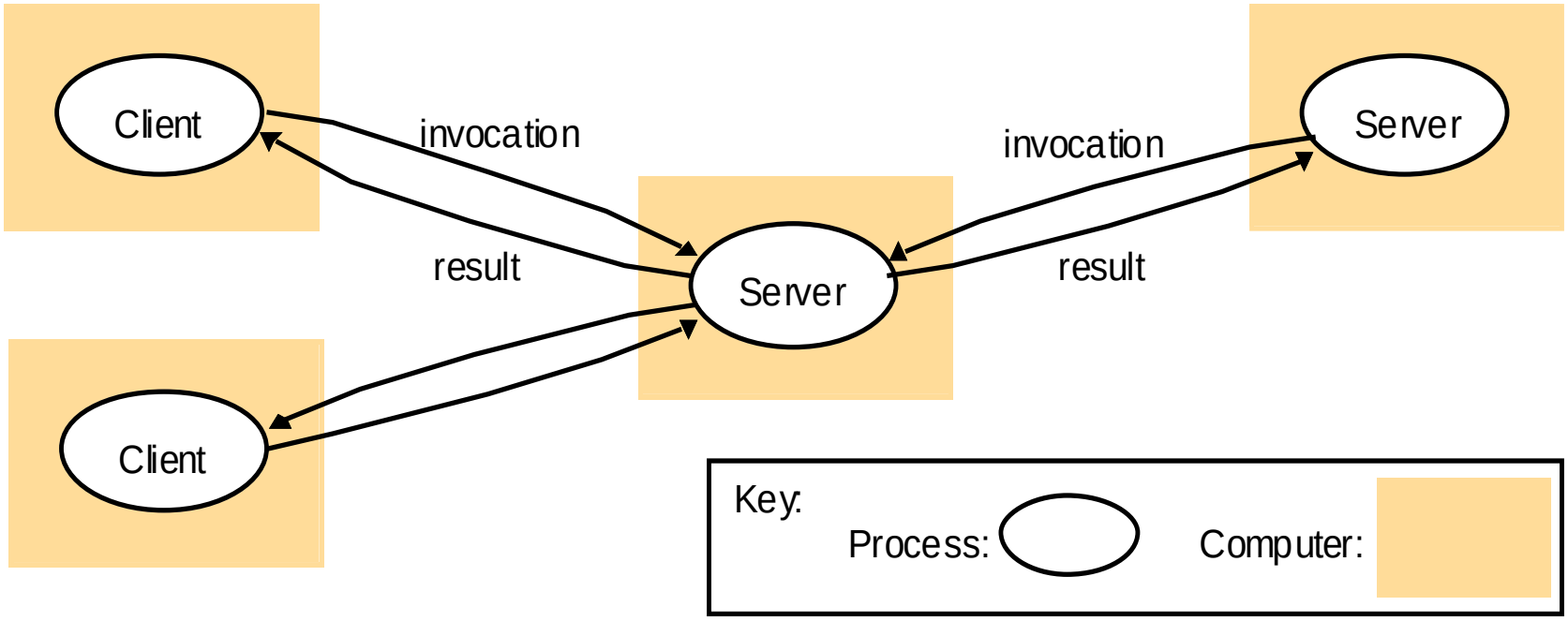
Application Layering

- **Traditional three-layered view**
  - User-interface layer contains units for an application's user interface
  - Processing layer contains the functions of an application, i.e. without specific data
  - Data layer contains the data that a client wants to manipulate through the application components
- This layering is found in many distributed information systems, using traditional database technology and accompanying applications.

Processing Level

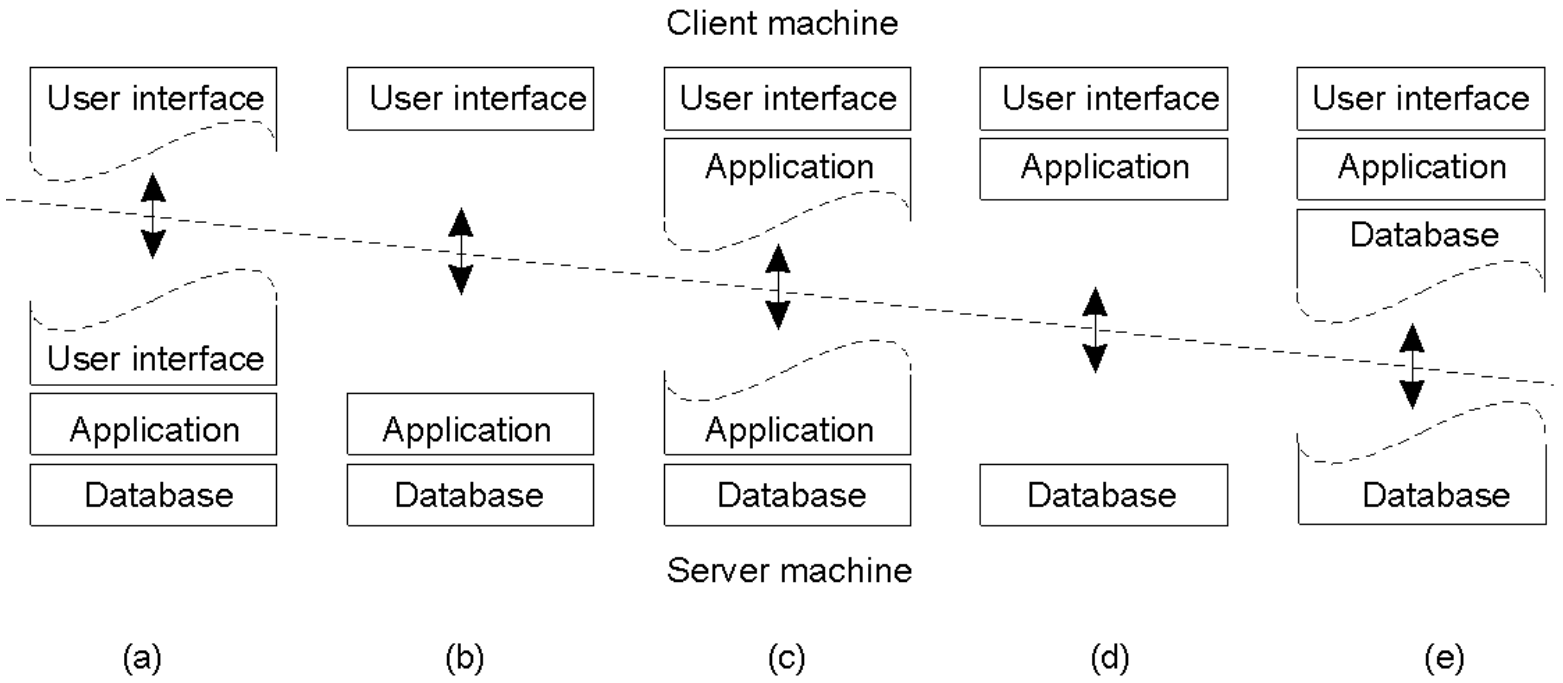


Clients Invoke Individual Servers



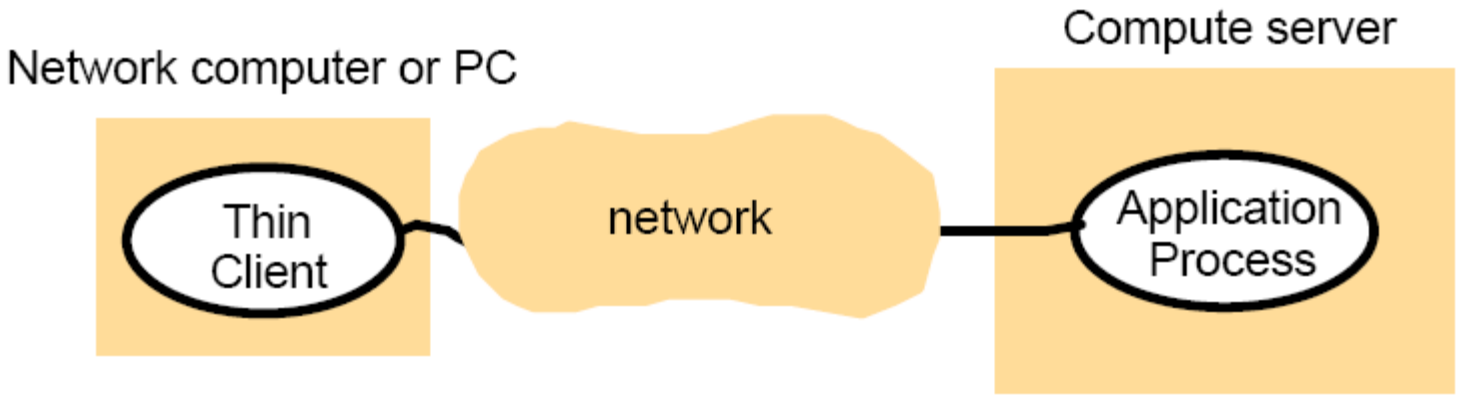
Multitiered Architectures (1)

- Single-tiered: dumb terminal/mainframe configuration
- Two-tiered: client/single server configuration
- Three-tiered: each layer on separate machine



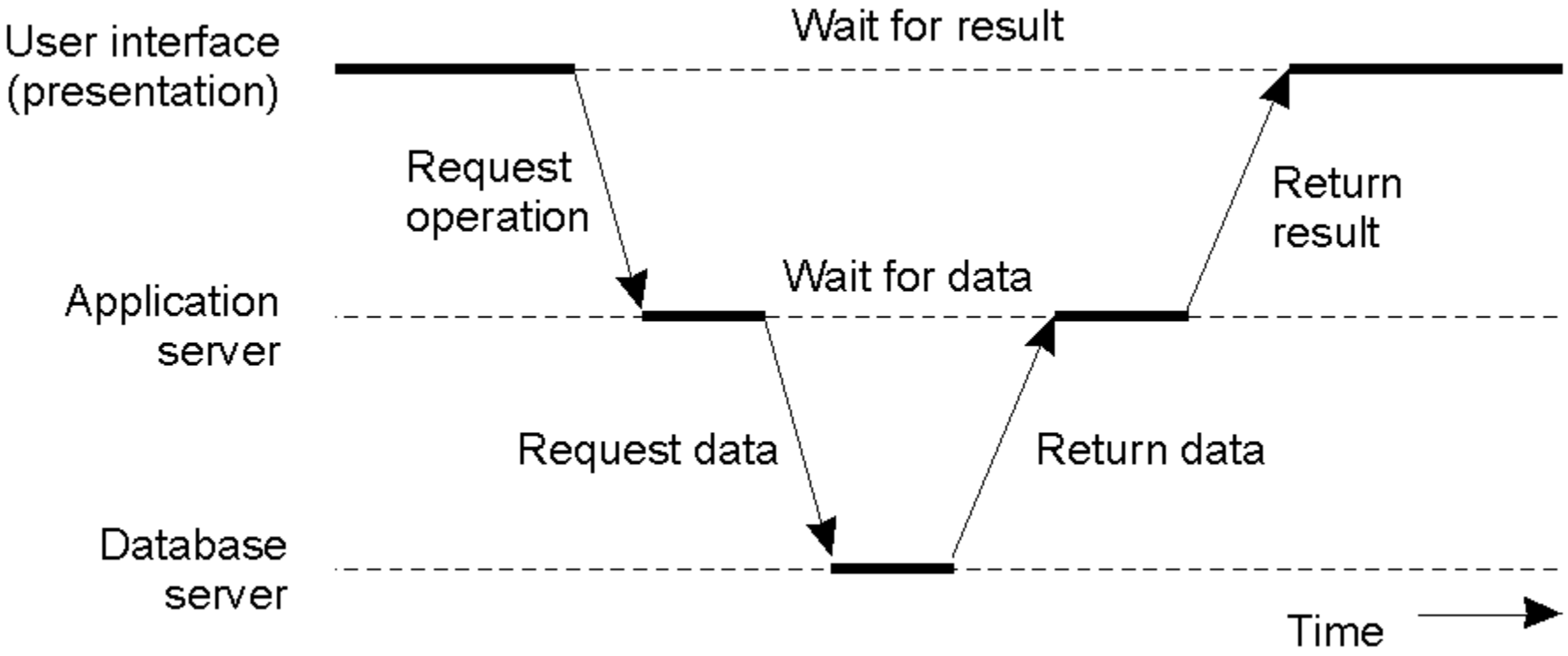
"Thin" Clients

- Thin Clients and Compute Servers
  - Executing graphical user interface on local computer while application executes on compute server
  - Example: X11 server (run on the application client side)
  - In reality: Palm Pilots, Mobile phones



Multitiered Architectures (2)

- An example of a server acting as a client.

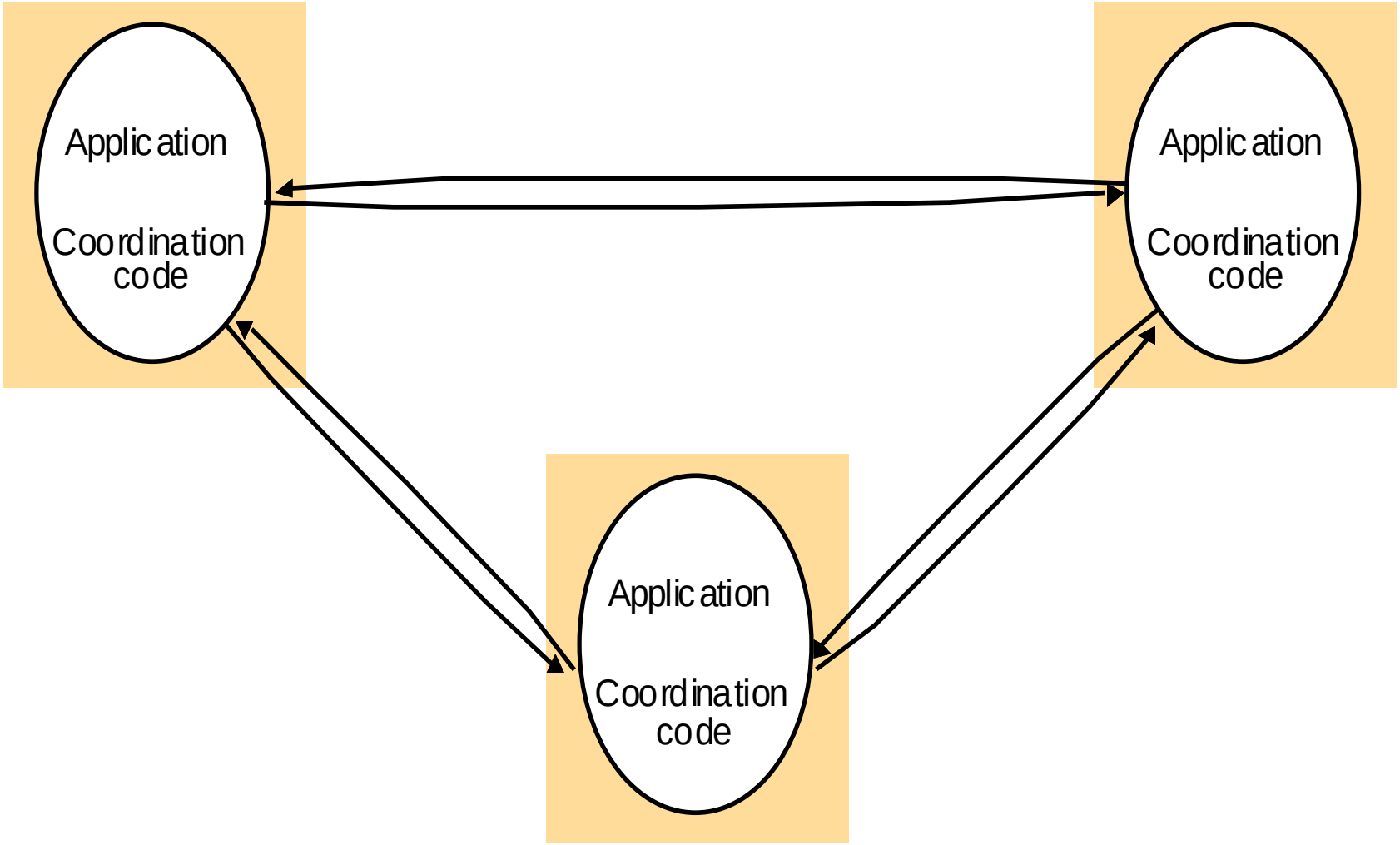


Decentralized Architecture

- Structured P2P: nodes are organized following a specific distributed data structure
- Unstructured P2P: nodes have randomly selected neighbors
- Hybrid P2P: some nodes are appointed special functions in a well-organized fashion

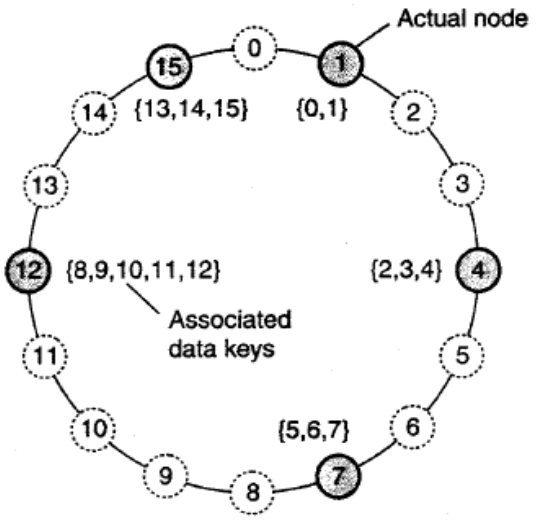
- In virtually all cases, we are dealing with overlay networks: data is routed over connections setup between the nodes (cf. application-level multicasting)

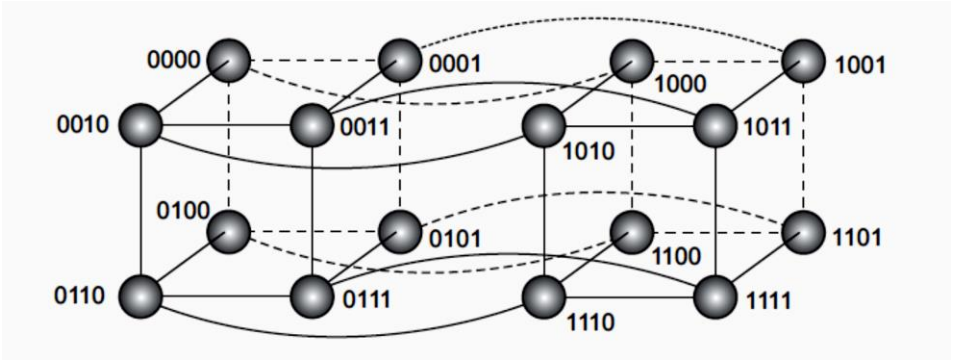
A Distributed Application based on Peer Processes



Structured P2P Systems

- Organize the nodes in a structured overlay network such as a logical ring, or a hypercube, and make specific nodes responsible for services based only on their ID.



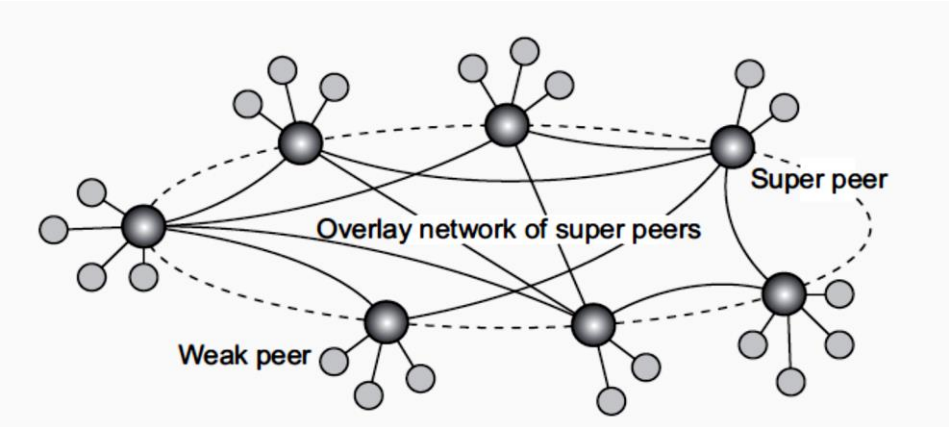


Unstructured P2P Systems

- Many unstructured P2P systems are organized as a random overlay: two nodes are linked with probability *p*.
- We can no longer look up information deterministically, but will have to resort to searching:
  - Flooding: node *u* sends a lookup query to all of its neighbors. A neighbor responds, or forwards (floods) the request. There are many variations:
    - Limited flooding (maximal number of forwarding)
    - Probabilistic flooding (flood only with a certain probability).
- Random walk: Randomly select a neighbor v. If v has the answer, it replies, otherwise v randomly selects one of its neighbors. Variation: parallel random walk. Works well with replicated data.

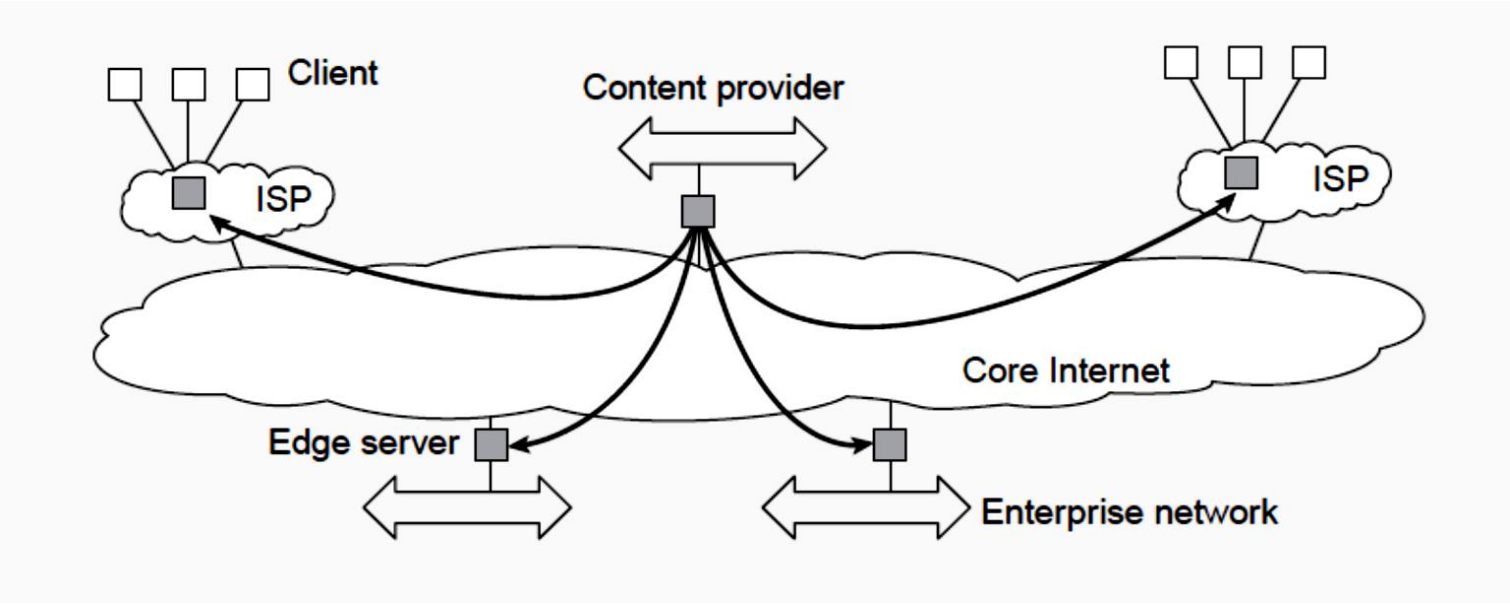
Superpeers

- Sometimes it helps to select a few nodes to do specific work: superpeer.
  - Peers maintaining an index (for search)
  - Peers monitoring the state of the network
  - Peers being able to setup connections



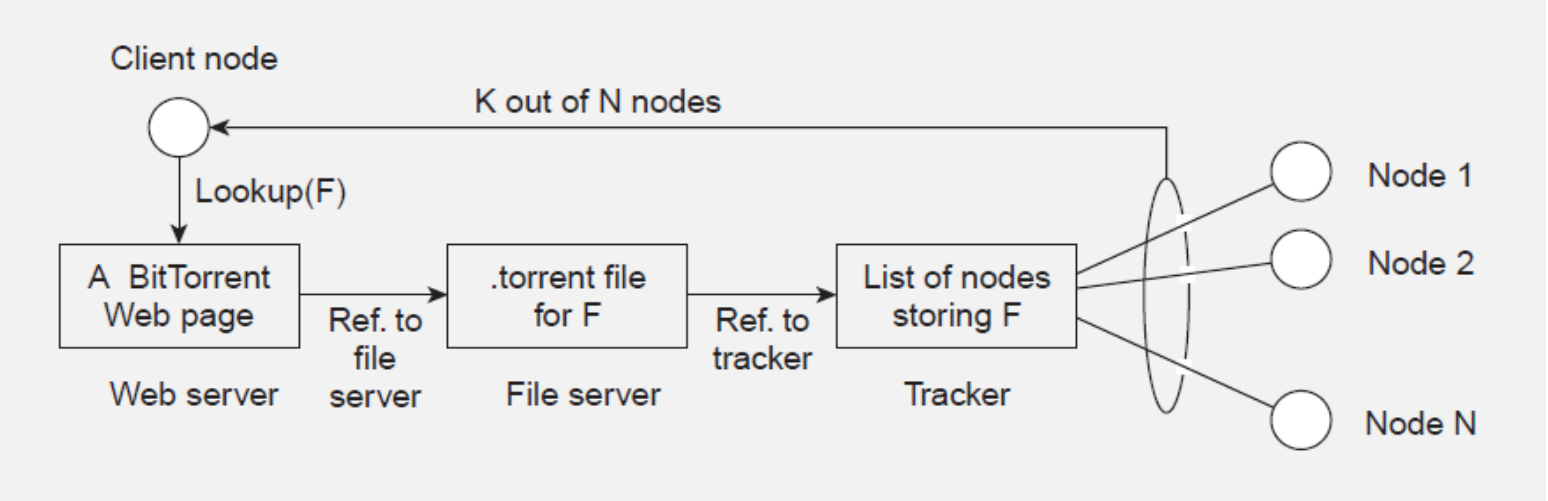
Hybrid Architectures

- Client-server combined with P2P
- Edge-server architectures, which are often used for Content Delivery Networks.



BitTorrent

- Once a node has identified where to download a file from, it joins a swarm of downloaders who in parallel get file chunks from the source, but also distribute these chunks amongst each other.

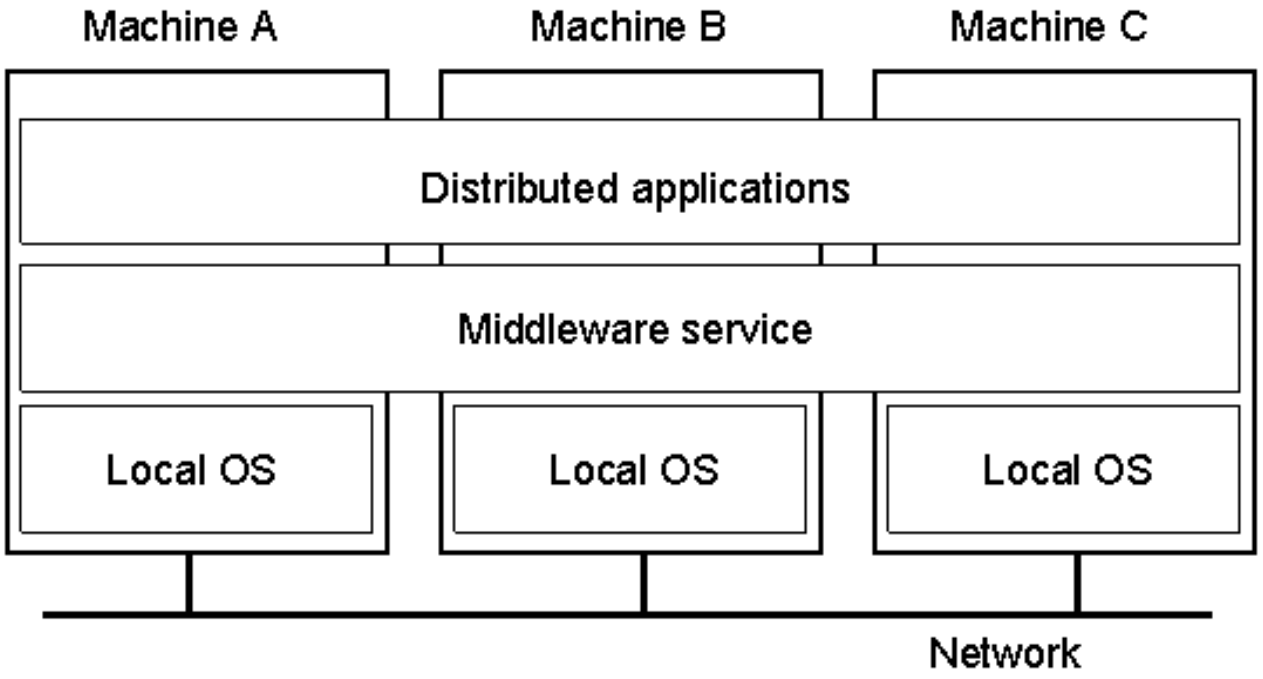


Middleware

- In many cases, distributed systems/applications are developed according to a specific architectural style. The chosen style may not be optimal in all cases → need to (dynamically) adapt the behavior of the middleware.

- **Interceptors**
  - Intercept the usual flow of control when invoking a remote object.

A distributed system organized as middleware



Self-managing Distributed Systems

- Distinction between system and software architectures blurs when automatic adaptively needs to be taken into account:
  - Self-configuration
  - Self-managing
  - Self-healing
  - Self-optimizing
  - …

There is a lot of hype going on in this field of autonomic computing.

Feedback Control Model

- In many cases, self-* systems are organized as a feedback control system.